Representing and Manipulating Floating Point

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Representing Floating Points

- **IEEE standard 754**
  - Established in 1985 as uniform standard for floating point arithmetic
    - Before that, many idiosyncratic formats
  - Supported by all major CPUs
  - William Kahan, a primary architect of IEEE 754, won the Turing Award in 1989.
  - Driven by numerical concerns
    - Nice standards for rounding, overflow, underflow
    - Hard to make go fast
    - Numerical analysts predominated over hardware types in defining standard.
### Fractional Binary Numbers (1)

**Representation**
- Bits to right of “binary point” represent fractional powers of 2
- Represents rational number: \( \sum_{k=-j}^{i} b_k \cdot 2^k \)
Fractional Binary Numbers (2)

- **Examples:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-3/4</td>
<td>101.11₂</td>
</tr>
<tr>
<td>2-7/8</td>
<td>10.111₂</td>
</tr>
<tr>
<td>63/64</td>
<td>0.111111₂</td>
</tr>
</tbody>
</table>

- **Observations**
  - Divide by 2 by shifting right
  - Multiply by 2 by shifting left
  - Numbers of form $0.111111\ldots₂$ just below $1.0$
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - Use notation $1.0 - \varepsilon$
Fractional Binary Numbers (3)

Representable numbers
- Can only exactly represent numbers of the form $x/2^k$
- Other numbers have repeating bit representations

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1/3$</td>
<td>0.010101010101[01]...$_2$</td>
</tr>
<tr>
<td>$1/5$</td>
<td>0.001100110011[0011]...$_2$</td>
</tr>
<tr>
<td>$1/10$</td>
<td>0.0001100110011[0011]...$_2$</td>
</tr>
</tbody>
</table>
FP Representation

- **Numerical form**: $-1^s \times M \times 2^E$
  - Sign bit $s$ determines whether number is negative or positive
  - Significand $M$ normally a fractional value in range $[1.0,2.0)$
  - Exponent $E$ weights value by power of two

- **Encoding**
  
<table>
<thead>
<tr>
<th>$s$</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
</table>

  - MSB is sign bit
  - $\text{exp}$ field encodes $E$ (Exponent)
  - $\text{frac}$ field encodes $M$ (Mantissa)
FP Precisions

**Encoding**

- MSB is sign bit
- exp field encodes $E$ (Exponent)
- frac field encodes $M$ (Mantissa)

**Sizes**

- Single precision: 8 exp bits, 23 frac bits (32bits total)
- Double precision: 11 exp bits, 52 frac bits (64bits total)
- Extended precision: 15 exp bits, 63 frac bits
  - Only found in Intel-compatible machines
  - Stored in 80 bits (1 bit wasted)
Normalized Values (1)

- **Condition:** $\exp \neq 000...0$ and $\exp \neq 111...1$

- **Exponent coded as biased value**
  - $E = \text{Exp} - \text{Bias}$
  - $\text{Exp}$: unsigned value denoted by $\exp$
  - $\text{Bias}$: Bias value
    - Single precision: 127 ($\text{Exp} : 1..254$, $E : -126..127$)
    - Double precision: 1023 ($\text{Exp} : 1..2046$, $E : -1022..1023$)

- **Significand coded with implied leading 1**
  - $M = 1.xxx...x_2$
    - Minimum when 000...0 ($M = 1.0$)
    - Maximum when 111...1 ($M = 2.0 - \varepsilon$)
  - Get extra leading bit for “free”
Normalized Values (2)

- **Value:** \( \text{float } f = 2003.0; \)
  \( 2003_{10} = 11111010011_2 = 1.1111010011_2 \times 2^{10} \)

- **Significand**
  \( M = 1.1111010011_2 \)
  \( \text{frac} = 1111010011000000000000000_2 \)

- **Exponent**
  \( E = 10 \)
  \( \text{Exp} = E + \text{Bias} = 10 + 127 = 137 = 10001001_2 \)

Floating Point Representation:
- **Hex:** 4 4 F A 6 0 0 0 0
- **Binary:** 0100 0100 1111 1010 0110 0000 0000 0000
- **137:** 100 0100 1
- **2003:** 1111 1010 0110
**Denormalized Values**

- **Condition:** $\text{exp} = 000...0$
- **Value**
  - Exponent value $E = 1 - \text{Bias}$
  - Significand value $M = 0.xxx...x_2$ (no implied leading 1)
- **Cases**
  - $\text{exp} = 000...0$, $\text{frac} = 000...0$
    - Represents value 0
    - Note that there are distinct values $+0$ and $-0$
  - $\text{exp} = 000...0$, $\text{frac} \neq 000...0$
    - Numbers very close to 0.0
    - “Gradual underflow”: possible numeric values are spaced evenly near 0.0
Special Values

- **Condition:** $\exp = 111...1$

- **Cases**
  - $\exp = 111...1, \frac{\text{frac}}{000...0}$
    - Represents value $\infty$ (infinity)
    - Operation that overflows
    - Both positive and negative
    - e.g. $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$
  - $\exp = 111...1, \frac{\text{frac}}{\neq 000...0}$
    - Not-a-Number (NaN)
    - Represents case when no numeric value can be determined
    - e.g., $\sqrt{-1}$, $\infty - \infty$, $\infty \times 0$, ...
Tiny FP Example (1)

- **8-bit floating point representation**
  - The sign bit is in the most significant bit
  - The next four bits are the \( \text{exp} \), with a bias of 7
  - The last three bits are the \( \text{frac} \)

- **Same general form as IEEE format**
  - Normalized, denormalized
  - Representation of 0, NaN, infinity

7 6 3 2 0

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>exp</th>
<th></th>
<th>frac</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Tiny FP Example (2)

**Values related to the exponent (Bias = 7)**

<table>
<thead>
<tr>
<th>Description</th>
<th>Exp</th>
<th>exp</th>
<th>E = Exp - Bias</th>
<th>$2^E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denormalized</td>
<td>0</td>
<td>0000</td>
<td>-6</td>
<td>1/64</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0001</td>
<td>-6</td>
<td>1/64</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0010</td>
<td>-5</td>
<td>1/32</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0011</td>
<td>-4</td>
<td>1/16</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0100</td>
<td>-3</td>
<td>1/8</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0101</td>
<td>-2</td>
<td>1/4</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0110</td>
<td>-1</td>
<td>1/2</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0111</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1000</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>1001</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1010</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>1011</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>1100</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>1101</td>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>1110</td>
<td>7</td>
<td>128</td>
</tr>
<tr>
<td>inf, NaN</td>
<td>15</td>
<td>1111</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Tiny FP Example (3)

### Dynamic range

<table>
<thead>
<tr>
<th>Description</th>
<th>Bit representation</th>
<th>e</th>
<th>E</th>
<th>f</th>
<th>M</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>0 0000 000</td>
<td>0</td>
<td>-6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Smallest pos.</td>
<td>0 0000 001</td>
<td>0</td>
<td>-6</td>
<td>1/8</td>
<td>1/8</td>
<td>1/512</td>
</tr>
<tr>
<td></td>
<td>0 0000 010</td>
<td>0</td>
<td>-6</td>
<td>2/8</td>
<td>2/8</td>
<td>2/512</td>
</tr>
<tr>
<td></td>
<td>0 0000 011</td>
<td>0</td>
<td>-6</td>
<td>3/8</td>
<td>3/8</td>
<td>3/512</td>
</tr>
<tr>
<td></td>
<td>0 0000 110</td>
<td>0</td>
<td>-6</td>
<td>6/8</td>
<td>6/8</td>
<td>6/512</td>
</tr>
<tr>
<td>Largest denorm.</td>
<td>0 0000 111</td>
<td>0</td>
<td>-6</td>
<td>7/8</td>
<td>7/8</td>
<td>7/512</td>
</tr>
<tr>
<td>Smallest norm.</td>
<td>0 0001 000</td>
<td>1</td>
<td>-6</td>
<td>0</td>
<td>8/8</td>
<td>8/512</td>
</tr>
<tr>
<td></td>
<td>0 0001 001</td>
<td>1</td>
<td>-6</td>
<td>1/8</td>
<td>9/8</td>
<td>9/512</td>
</tr>
<tr>
<td></td>
<td>0 0110 110</td>
<td>6</td>
<td>-1</td>
<td>6/8</td>
<td>14/8</td>
<td>14/16</td>
</tr>
<tr>
<td></td>
<td>0 0110 111</td>
<td>6</td>
<td>-1</td>
<td>7/8</td>
<td>15/8</td>
<td>15/16</td>
</tr>
<tr>
<td></td>
<td>0 0111 000</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>8/8</td>
<td>1</td>
</tr>
<tr>
<td>One</td>
<td>0 0111 001</td>
<td>7</td>
<td>0</td>
<td>1/8</td>
<td>9/8</td>
<td>9/8</td>
</tr>
<tr>
<td></td>
<td>0 0111 010</td>
<td>7</td>
<td>0</td>
<td>2/8</td>
<td>10/8</td>
<td>10/8</td>
</tr>
<tr>
<td></td>
<td>0 1110 110</td>
<td>14</td>
<td>7</td>
<td>6/8</td>
<td>14/8</td>
<td>224</td>
</tr>
<tr>
<td>Largest norm.</td>
<td>0 1110 111</td>
<td>14</td>
<td>7</td>
<td>7/8</td>
<td>15/8</td>
<td>240</td>
</tr>
<tr>
<td>Infinity</td>
<td>0 1111 000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+∞</td>
</tr>
</tbody>
</table>
Tiny FP Example (4)

**Encoded values (nonnegative numbers only)**

- **0 1101 XXX = \((8/8 \sim 15/8) \times 2^6\)**
- **0 1110 XXX = \((8/8 \sim 15/8) \times 2^7\)**

- **0 0111 XXX = \((8/8 \sim 15/8) \times 2^0\)**
- **0 1000 XXX = \((8/8 \sim 15/8) \times 2^1\)**

- **0 0000 XXX = \((0/8 \sim 7/8) \times 2^{-6}\)**
- **0 0001 XXX = \((8/8 \sim 15/8) \times 2^{-6}\)**

- **0 0011 XXX = \((8/8 \sim 15/8) \times 2^{-4}\)**

*With denormalization*

- **0 0000 XXX = \((8/8 \sim 15/8) \times 2^{-7}\)**
# Interesting Numbers

<table>
<thead>
<tr>
<th>Description</th>
<th>exp</th>
<th>frac</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>000 ... 00</td>
<td>000 ... 00</td>
<td>0.0</td>
</tr>
<tr>
<td>Smallest Positive denormalized</td>
<td>000 ... 00</td>
<td>000 ... 01</td>
<td>Single: $2^{-23} X 2^{-126} \approx 1.4 \times 10^{-45}$ Double: $2^{-52} X 2^{-1022} \approx 4.9 \times 10^{-324}$</td>
</tr>
<tr>
<td>Largest Denormalized</td>
<td>000 ... 00</td>
<td>111 ... 11</td>
<td>Single: $(1.0 - \varepsilon) X 2^{-126} \approx 1.18 \times 10^{-38}$ Double: $(1.0 - \varepsilon) X 2^{-1022} \approx 2.2 \times 10^{-308}$</td>
</tr>
<tr>
<td>Smallest Positive Normalized</td>
<td>000 ... 01</td>
<td>000 ... 00</td>
<td>Single: $1.0 X 2^{-126}$, Double: $1.0 X 2^{-1022}$ (Just larger than largest denormalized)</td>
</tr>
<tr>
<td>One</td>
<td>011 ... 11</td>
<td>000 ... 00</td>
<td>1.0</td>
</tr>
<tr>
<td>Largest Normalized</td>
<td>111 ... 10</td>
<td>111 ... 11</td>
<td>Single: $(2.0 - \varepsilon) X 2^{127} \approx 3.4 \times 10^{38}$ Double: $(2.0 - \varepsilon) X 2^{1023} \approx 1.8 \times 10^{308}$</td>
</tr>
</tbody>
</table>
Special Properties

- **FP zero same as integer zero**
  - All bits = 0

- **Can (almost) use unsigned integer comparison**
  - Must first compare sign bits
  - Must consider -0 = 0
  - NaNs problematic
    - Will be greater than any other values
  - Otherwise OK
    - Denorm vs. normalized
    - Normalized vs. Infinity
Floating Point in C (1)

- C guarantees two levels
  - float (single precision) vs. double (double precision)

- Conversions
  - double or float → int
    - Truncates fractional part
    - Like rounding toward zero
    - Not defined when out of range or NaN
      » Generally sets to Tmin
  - int → double
    - Exact conversion, as long as int has ≤ 53 bit word size
  - int → float
    - Will round according to rounding mode
Floating Point in C (2)

- **Example 1:**

```c
#include <stdio.h>

int main () {
    int n = 123456789;
    int nf, ng;
    float f;
    double g;

    f = (float) n;
    g = (double) n;
    nf = (int) f;
    ng = (int) g;
    printf ("nf=%d ng=%d\n", nf, ng);
}
```
Example 2:

```c
#include <stdio.h>

int main () {
    double d;
    
    d = 1.0 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
        + 0.1 + 0.1 + 0.1 + 0.1 + 0.1;
    
    printf ("d = %.20f\n", d);
}
```
Example 3:

```c
#include <stdio.h>

int main () {
    float f1 = (3.14 + 1e20) – 1e20;
    float f2 = 3.14 + (1e20 – 1e20);

    printf (“f1 = %f, f2 = %f\n”, f1, f2);
}
```
Ariane 5

- Ariane 5 tragedy (June 4, 1996)
  - Exploded 37 seconds after liftoff
  - Satellites worth $500 million

- Why?
  - Computed horizontal velocity as floating point number
  - Converted to 16-bit integer
    - Careful analysis of Ariane 4 trajectory proved 16-bit is enough
  - Reused a module from 10-year-old s/w
    - Overflowed for Ariane 5
    - No precise specification for the S/W
IEEE floating point has clear mathematical properties

- Represents numbers of form \( M \times 2^E \)
- Can reason about operations independent of implementation
  - As if computed with perfect precision and then rounded
- Not the same as real arithmetic
  - Violates associativity/distributivity
  - Makes life difficult for compilers and serious numerical applications programmers
Main Points

- It’s all about bits & bytes
  - Numbers, programs, text, ...

- Different machines follow different conventions
  - Word size
  - Byte ordering
  - Representations (Integer, Floating-Point)

- When programming, be aware of
  - Type casting & mixed signed/unsigned expressions
  - Overflow
  - Error propagation
  - Byte ordering